A2

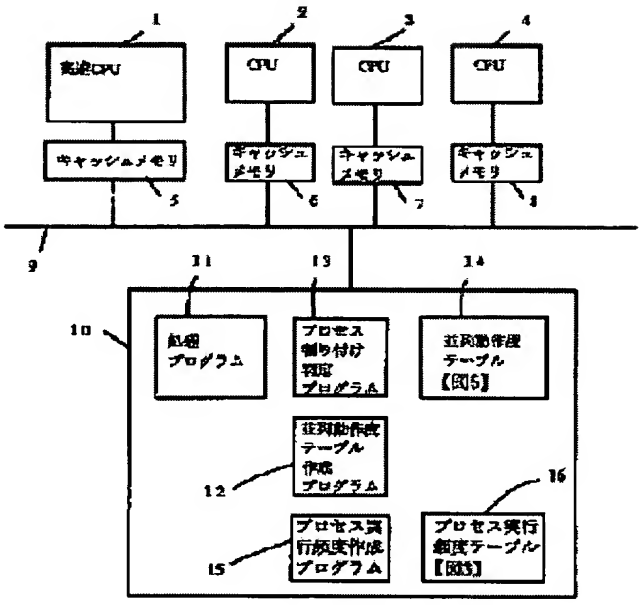
MULTIPROCESSOR SYSTEM

Patent number: JP10143380
Publication date: 1998-05-29
Inventor: FUNAKOSHI KAZUMI; TAMURA SHIGERU
Applicant: HITACHI LTD;; HITACHI INF & CONTROL SYST INC
Classification:
- international: G06F9/46; G06F15/16
- european:
Application number: JP19960295185 19961107
Priority number(s):

Abstract of **JP10143380**

PROBLEM TO BE SOLVED: To easily perform parallel processing between processes by using a conventional sequential-type language program at it is by performing distribution to respective processor of the processes according to the parallel operation ability of the processes forming a program and the length of processing time.

SOLUTION: A parallel operation ability table 14 is formed prior to an operation of a processing program 11. A process allocation judgment program 13 refers to the parallel operation ability table 14 and allocates a process forming the processing program 11 to multi-CPUs 1 to 4. After completion of the allocation, the allocated process is made to correspond to cash memories 5 to 8 of the destination of the allocation, and transmitted and stored therein. Then, according to the result of the allocation, sequential processing and parallel processing are performed by the operation of the multi-CPUs 1 to 4 to execute the program 11. Thus, it is effectively distributed to two types of hard resources, a high-speed processor and plural processors for parallel processing, and a totally efficient multiprocessor is formed.



(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平10－143380

(43)公開日　平成10年(1998)５月29日

(51)Int.Cl.[6]		識別記号	FI		
Ｇ０６Ｆ	9/46	３６０	Ｇ０６Ｆ	9/46	３６０Ｂ
	15/16	３７０		15/16	３７０Ｎ

審査請求　未請求　請求項の数６　ＯＬ　（全 10 頁）

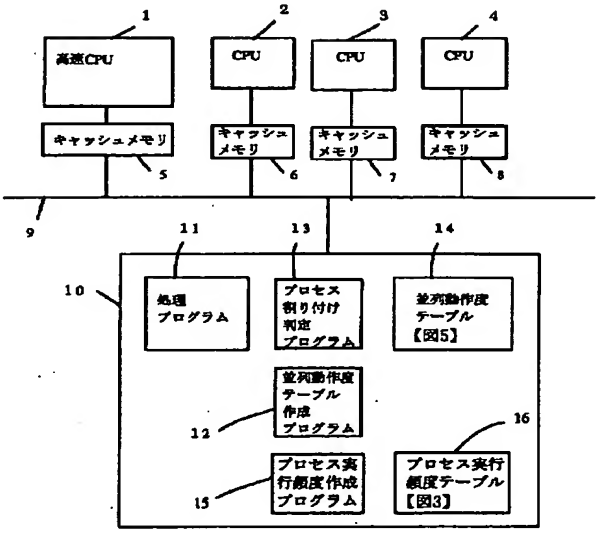
(21)出願番号　特願平8－295185

(22)出願日　平成８年(1996)11月７日

(71)出願人　000005108
株式会社日立製作所
東京都千代田区神田駿河台四丁目６番地
(71)出願人　000153443
株式会社日立情報制御システム
茨城県日立市大みか町５丁目２番１号
(72)発明者　舟越　和己
茨城県日立市大みか町五丁目２番１号　株式会社日立情報制御システム内
(72)発明者　田村　滋
茨城県日立市大みか町五丁目２番１号　株式会社日立製作所大みか工場内
(74)代理人　弁理士　高崎　芳紘

(54)【発明の名称】　マルチプロセッサシステム

(57)【要約】
【課題】　逐次型プログラムを高速ＣＰＵと低速ＣＰＵとに割り付けて、効率的な処理をはかりたい。
【解決手段】　１台の高速ＣＰＵ１と複数台の低速ＣＰＵ２、３、４と、共通バス９、共有メモリ１０とより成るマルチプロセッサシステムより成る。高速ＣＰＵ１と低速ＣＰＵ２、３、４とに処理プログラム１１のプロセスＰ1～Ｐnを並列動作度の大小及び処理時間の大小に応じて割り付ける。



1

【特許請求の範囲】

【請求項１】　逐次処理用の高速プロセッサと並列処理用の複数のプロセッサを持ち、プログラムを構成するプロセスの並列動作度及び処理時間の大小によりプロセスの各プロセッサへの配分を行うことを特徴とするマルチプロセッサシステム。

【請求項２】　逐次処理用の高速プロセッサと並列処理用の複数のプロセッサを持ち、プログラムを構成するプロセスの並列動作度及び処理時間をオフラインで求めてメモリに登録しておき、プログラム実行時に上記プロセスの並列動作度及び処理時間の大小により、プロセスの各プロセッサへの配分を行うものとしたことを特徴とするマルチプロセッサシステム。

【請求項３】　請求項１又は２において、プログラムを構成するプロセスの並列動作度及び処理時間は、すべてのプロセスの対について、２つのプロセッサで同時実行して求めた並列処理可能度及び実行頻度とから得るものとしたマルチプロセッサシステム。

【請求項４】　逐次処理用の高速プロセッサと並列処理用の複数のプロセッサを持つマルチプロセッサシステムにおいて、逐次型言語で作成されたプログラムについて、そのプログラムを構成するプロセスの並列動作度及び処理時間の大小により、プロセスの各プロセッサへの配分の割り付けを行うマルチプロセッサシステム。

【請求項５】　逐次処理用の高速プロセッサと並列処理用の複数のプロセッサとを持ち、逐次型言語で作成されたプログラムを実行するマルチプロセッサシステムにおいて、

プログラムを構成するプロセスについて求めた並列動作度及び処理時間をメモリに格納しておき、上記プログラムのプロセス実行待ちキューの中の先頭のプロセスについて、メモリに格納してあるプロセスの並列動作度から得られる並列度の大小を比較し、小であれば高速プロセッサが空くのを待って当該プロセスを高速プロセッサに割り付け、大であればプロセッサ数の並列度大のプロセスをキューより取り出しその各プロセッサについての上記メモリに格納してある処理時間を参照して実行させるべき並列度大のプロセスを処理時間順に並べ、全プロセッサが空くのを待って、処理時間が最長のプロセスを高速プロセッサに割り当て、残りを並列プロセッサに割り当てるものとしたマルチプロセッサシステム。

【請求項６】　高速プロセッサと、複数の低速プロセッサと、各プロセッサ対応のキャッシュメモリと、キャッシュメモリに接続された共通バスと、この共通バスに接続された共有メモリと、を有すると共に、共通メモリ上でプログラムを構成するプロセスの並列化の度合いを示すテーブルをオフライン的に作成し、このプロセスの並列度の度合い及び処理時間の大小によりプロセスをプロセッサに割り当て対応キャッシュメモリにその旨の連絡及び対応データを送出するものとしたマルチプロセッサシステム。

2

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、電子計算機に係わり、特に複数のプロセッサを備え１つの演算処理を並列に高速処理するマルチプロセッサシステムに関する。

【０００２】

【従来の技術】マルチプロセッサシステムは、逐次型言語（例えばFORTRANやC言語）を用いて作成されたプログラムに対し、自動並列化コンパイラ等により並列処理可能な処理を決め、プログラムを実行している。例えば１つのプログラム中に繰り返し処理であるＤＯループが存在する場合、このＤＯループを複数のＣＰＵで処理している。例えば、処理プログラム中に

ＤＯ　　Ｉ＝１、４０

Ａ（Ｉ）　＝Ｂ（Ｉ）＋Ｃ（Ｉ）

なる演算がある場合、Ｉ＝１～１０、Ｉ＝１～２０、Ｉ＝２１～３０、Ｉ＝３１～４０の演算を各々別のＣＰＵ（例えば別の４つのＣＰＵ）に割り当て、並列処理している。

【０００３】また、さらに進んだ方式では、最初から並列処理用言語を使用して複数プロセスの並列性記述によりプログラムを作成し、自動並列化コンパイラで対応困難なプロセス間の並列処理を実現している。並列処理用言語は、例えば、「並列処理技術、１９９１年、コロナ社発行」の第１０５頁から第１１３頁において論じられているように、並列実行可能プロセスの記述等をコンパイラだけに頼らずに積極的にプログラマの知見を利用し効率良い並列処理を実現しようとするアプローチである。前記で作成されたプロセスをマルチプロセッサへ割り付ける方法としては、従来のマルチプロセッサシステムでは、複数あるＣＰＵの稼働率を高めるため、逐次処理を行うＣＰＵや並列処理を行うＣＰＵを常にダイナミックに変えている。

【０００４】また、別の従来のマルチプロセッサシステムでは、ある１つのプログラムを実行中だけ、逐次処理ならばｎ個中の任意の１個のＣＰＵに固定し、並列処理ならば並列処理を行うＣＰＵを固定的に割り当て、逐次処理と並列処理が動作するタイミングをフラグによりとる（逐次処理実行中は並列処理は動作せず、並列処理が全て終了するまでは次の逐次処理は動作しないということをフラグにより管理）ことにより、ハード量が少なくオーバーヘッドの小さいマルチプロセッサシステムを実現している。ここで、マルチプロセッサは、同じ性能のｎ個のプロセッサからなっている。この従来例としては、特開平２－１４４６５７号がある。

【０００５】

【発明が解決しようとする課題】まず第１に並列可能な処理の決定について、上記従来技術の自動並列化コンパイラは、ＤＯループの並列処理化には有効であるが、プ

ロセス間の並列処理化に対しては適していない。これに対しては、並列処理用言語によるプログラミングがあるが、この方法では従来の逐次型言語のプログラムをそのままでは使用できない。次に、プロセスをＣＰＵへ配分する方式については、上記従来技術のように、逐次処理を行うＣＰＵや並列処理を行うＣＰＵを常にダイナミックに変えるように構成すると、ＣＰＵの稼働率は高くなるが、ダイナミックに変動させるためのハード量が増大してしまい、また各ＣＰＵの同期用のハード量も増大してしまうという問題が生じる。さらに、オーバーヘッドが大きくなるという不都合も生じる。

【０００６】また、プログラムを実行する間だけそれを行うＣＰＵを固定する方法は、オーバーヘッドは少ないが、実行するＣＰＵの選択は空いているＣＰＵが選ばれるのみなので、以下の理由により当該プログラムを実行するのに適切なＣＰＵ資源が選ばれるとは限らない。プログラムで対象とする通常の業務処理は、逐次処理と並列処理が混在している場合が一般的である。例えば、電力系統の長期計画業務（例：年間計画）では、期間内の潮流図を日付を変えて繰り返し作成する処理（並列可能な処理）と、この結果に対して系統解析を行う処理（逐次処理）とに分けられる。この場合、逐次処理は１台のＣＰＵで実行されるので出来るだけ高速なＣＰＵがよい。一方において、並列処理は、複数ＣＰＵで実行されるので高速でなくても台数効果が出せる。従って、従来の同じ性能のマルチプロセッサ構成よりもＣＰＵの性能を逐次用の高速プロセッサと並列用の複数のプロセッサにしたマルチプロセッサ構成の方が有効である。

【０００７】本発明の目的は、従来の逐次型言語のプログラムをそのまま使用してプロセス間の並列処理を容易に実現するマルチプロセッサシステムを提供するものである。さらに本発明の目的は、逐次処理と並列処理が混在するプログラムを逐次用の高速プロセッサと並列用の複数のプロセッサという２種類のハード資源に有効に配分可能にして、トータルとして効率的なプロセッサ管理や効率的なプログラム処理の実現をはかるマルチプロセッサシステムを提供することにある。

【０００８】

【課題を解決するための手段】本発明は、逐次処理用の高速プロセッサと並列処理用の複数のプロセッサを持ち、プログラムを構成するプロセスの並列動作度及び処理時間の大小によりプロセスの各プロセッサへの配分を行うことを特徴とするマルチプロセッサシステムを開示する。

【０００９】更に本発明は、逐次処理用の高速プロセッサと並列処理用の複数のプロセッサを持ち、プログラムを構成するプロセスの並列動作度及び処理時間をオフラインで求めてメモリに登録しておき、プログラム実行時に上記プロセスの並列動作度及び処理時間の大小により、プロセスの各プロセッサへの配分を行うものとした

ことを特徴とするマルチプロセッサシステムを開示する。

【００１０】さらに本発明は、プログラムを構成するプロセスの並列動作度及び処理時間は、すべてのプロセスの対について、２つのプロセッサで同時実行して求めた並列処理可能度及び実行頻度とから得るものとしたマルチプロセッサシステムを開示する。

【００１１】更に本発明は、逐次処理用の高速プロセッサと並列処理用の複数のプロセッサを持つマルチプロセッサシステムにおいて、逐次型言語で作成されたプログラムについて、そのプログラムを構成するプロセスの並列動作度及び処理時間の大小により、プロセスの各プロセッサへの配分の割り付けを行うマルチプロセッサシステムを開示する。

【００１２】更に本発明は、逐次処理用の高速プロセッサと並列処理用の複数のプロセッサとを持ち、逐次型言語で作成されたプログラムを実行するマルチプロセッサシステムにおいて、プログラムを構成するプロセスについて求めた並列動作度及び処理時間をメモリに格納しておき、上記プログラムのプロセス実行待ちキューの中の先頭のプロセスについて、メモリに格納してあるプロセスの並列動作度から得られる並列度の大小を比較し、小であれば高速プロセッサが空くのを待って当該プロセスを高速プロセッサに割り付け、大であればプロセッサ数の並列度大のプロセスをキューより取り出しその各プロセッサについての上記メモリに格納してある処理時間を参照して実行させるべき並列度大のプロセスを処理時間順に並べ、全プロセッサが空くのを待って、処理時間が最長のプロセスを高速プロセッサに割り当て、残りを並列プロセッサに割り当てるものとしたマルチプロセッサシステムを開示する。

【００１３】

【発明の実施の形態】以下、本発明の実施の形態を図面を参照して説明する。図１は、本発明の実施の形態に関するマルチプロセッサシステムの構成図である。本実施の形態のマルチプロセッサシステムは、逐次処理と並列処理を効率的に実行できるハード構成を目的として逐次処理用の高速プロセッサ（ＣＰＵ）１と並列処理用の複数のプロセッサ（ＣＰＵ）２～４から成る。高速ＣＰＵ１と３つの低速ＣＰＵ２、３、４は各々、キャッシュメモリ５、６、７、８を介して共通バス９に接続されている。低速ＣＰＵ２、３、４は同一機構（又は同一構成）より成るものとする。

【００１４】このバス９には、共有メモリ１０が接続されている。共有メモリ１０には、処理プログラム１１と並列動作度テーブル作成プログラム１２及びプロセス割り付けプログラム１３と並列動作度テーブル１４が格納されている。また、プロセス実行頻度の情報として、プロセス実行頻度作成プログラム１５とプロセス実行頻度テーブル１６が格納されている。

5

【００１５】ここで共通メモリ１０内のプログラム１１、１２、１３、１５、及びテーブル１４、１６について簡単に説明する。詳細は追って説明する。処理プログラム１１…マルチプロセッサシステムの処理対象とするプログラムのことであり、逐次型言語で作成されたプログラムを指す。このプログラム１１は複数のプロセスA_1～A_nから成る。プロセス割り付け判定プログラム１３…プロセスの並列動作度テーブル１４（図５の１４）を参照して、上記処理プログラムを構成するプロセスA_1～A_nについて、マルチプロセッサシステムのＣＰＵ１～４への割り付けを行う。ここで割り付けとは、該当のＣＰＵでプロセスを実行させることであり、プロセスのスケジューリング機能に相当する。並列動作度テーブル作成プログラム１２…プロセスの並列動作度テーブル１４（図５の１４）を事前に作成するためのプログラムである。並列動作度テーブル１４…並列動作度テーブル作成プログラム１２によって作られた並列動作度を、プロセス対応にテーブル化して記憶したものである。プロセス実行頻度テーブル１６…プロセス対応に、あるカウント時間内での実行回数を記録したテーブルである。このテーブル１６は、処理プログラム１０の実際の実行に先立って、得たものであり、並列動作度テーブル１４の作成のために使う。具体例としての図３のテーブル１６が該当する。プロセス実行頻度作成プログラム１５…プロセス実行頻度テーブル１６を作成するためのプログラムである。このプログラム１５は、処理プログラム１０の実際の実行に先立ち動作させる。

【００１６】次に、図１のマルチプロセッサシステムの全体動作を説明する。

（１）、並列動作度テーブル１４の作成…処理プログラム１１の動作に先立って並列動作度テーブル１４を作成する。この並列動作度テーブル１４は、２段階で作成されるものであり、先ずプロセス実行頻度作成プログラム１５によってプロセス実行頻度テーブル１６を作成し、次いで並列動作度テーブル作成プログラム１２によって、テーブル１６を利用して並列動作度テーブル１４を作成することになる。

（２）、処理プログラム１１を構成するプロセスのマルチＣＰＵ１～４への割り付け…プロセス割り付け判定プログラム１３が（１）で算出した並列動作度テーブル１４を参照して、マルチＣＰＵ１～４へ、処理プログラム１１を構成するプロセスA_1～A_nを割り付ける。割り付け終了後に、割り付けたプロセスを、割り付け先のキャッシュメモリ５～８に対応ずけて送り格納する。

（３）、処理プログラム１１の実行…（２）の割り付け結果に従って、マルチＣＰＵ１～４の動作により逐次処理、並びに並列処理がなされ、プログラム１１の実行をはかる。

【００１７】並列動作度作成プログラム１２の起動の時期は任意であり、オフライン的な処理で行う。また、図

6

１のマルチプロセッサシステムを利用して作成してもよく、図１とは異なる他のマルチプロセッサシステムによって作成しても良い。要は、図１のマルチプロセッサシステムとしての立ち上げ前に行えばよい。図２には、そうした一例を示し、図１のマルチプロセッサシステムを利用して並列動作度を自動作成する時の、並列動作度作成プログラム１２の動作時期と動作内容とを示す計算機モード（マルチプロセッサシステムとしての計算機モードの意味）図である。ここで、図２の中でフロー２４と２５とが、並列動作度作成プログラム１２である。

【００１８】図２において、計算機が立ち上げ処理２１を完了すると、フロー２２に移り、並列動作度テーブル作成モードに入るか否かが表示画面に表示され、操作者はこれをみて、並列動作度テーブル作成モードに入るか否かの指示を行う。並列動作度テーブル作成モードにすると、並列動作度テーブル１４の作成２４を行い、次いでフロー２５に移り、作成した並列動作度テーブル１４の内容を磁気ディスク等の外部記憶装置に保存する。ここで作成した並列動作度テーブル１４は、通常モード（フロー２６）においてプロセスの並列度判定に用いられる。フロー２２で、並列動作度テーブル作成モードにしなければ、外部記憶装置に保存されているすでに作成（フロー２４、２５によって）ずみの並列動作度テーブル１４を共有メモリ１０に展開し、通常モードの処理２６へ遷移する。通常モードでは、処理プログラム１１による計算機の本来の業務処理を行う。以上の２つのモードの終了は計算機停止処理２７により行う。並列度作成モードから通常モードへの切り替え（又はその逆）は、一旦、計算機を停止させる（フロー２７）が、並列動作度作成モードはしばしば行うものではないので、この方式で十分である。

【００１９】図３（ａ）は、プロセス実行頻度作成プログラム１５の処理フロー、図３（ｂ）はプロセス実行頻度テーブル１６を示す図である。処理プログラム１１のシミュレーションによる処理を行って各プロセスA_i（ｉ＝１～ｎ）についてプロセスが実行される毎にプロセスの実行回数をカウントし（フロー３１）、プロセス実行頻度テーブル１６の中味を更新する（フロー３２）。プロセス実行頻度テーブル１６は、プロセス毎の実行回数のカウント数ｒとカウント期間Ｗ（カウント開始時刻t_s及びカウント終了時刻t_eとの差分）から成る。カウント時間Ｗの期間中にそのプロセスA_iが何回実行されたかをカウント数ｒで示している。

【００２０】図４は、図２の並列動作度作成プログラム１２の中の処理フロー２４の詳細フローである。並列動作度テーブル作成モードでは、まず、フロー４１によって、プロセスA_1～A_nにおける２つのプロセスのすべての組合せのそれぞれについて、２つの同一機構（又は同一機種。例えば図１のＣＰＵ２と３）のプロセッサで同時に実行できる度合いを示す並列処理可能度λ_{ij}（ｉ＝

7

１～ｎ、ｊ＝１～ｎ、ｉ≠ｊ）を求める。このようにして任意の２つのプロセス並列処理可能度λijは、２つのプロセスを同時に２つのプロセッサでモニタランさせることにより求める。このようにして求めた並列処理可能度λijに対して図３（ｂ）のプロセスの実行頻度を考慮することにより、各プロセスをプロセッサ１～４へ配分する際の指標となる並列動作度λiを求める（フロー４２、４３）。図５（ａ）は並列処理可能度λij、及び図５（ｂ）は並列動作度λiを示す図である。

【００２１】次に、図５を用いて並列処理可能度λij（図５（ａ））及び並列動作度λi（図５（ｂ））について説明する。図５において、ユーザプログラム５のプロセスをA1、A2、…、Anとする。プロセスＡｉを単独に実行させた時の処理時間をｔiとする。またこれらのAi、Aj（１≦ｉ、ｊ≦ｎ）からなるプロセスの組合せ５２を考え、このプロセスAi、Ajを２つのプロセッサで同時に動作させる。この時のAiの処理時間をｔijとする。プロセスAiを単独に実行させた時の処理時間ｔiと、他のプロセスAjと同時に実行させた時の処理時間ｔijを比較した場合、その比ｔi／ｔijは１以下の値であり、この値が１に近いほど並列処理可能度が高いと考えられる。一般に他のプロセスと同時に実行したときは処理時間が長くなる（ｔi≦ｔij）ためである。そこで、プロセスAiのプロセスAjに対する並列処理可能度λijを次の式により定義する。

【数１】λij＝ｔi／ｔij

こうして求めた並列処理可能度λijが図５（ａ）の内容である。

【００２２】また、プロセスAjの実行頻度（単位時間内での実行回数）μjは、プロセス実行頻度テーブル３３の実行回数ｒjとカウント期間（ｔs～ｔe）より、下記で算出する。

【数２】μj＝ｒj／（ｔs～ｔe）

次に、プロセスAiをプロセッサへ配分する際の指標となる並列動作度λiは、他のプロセスが動作している時にプロセスAiが動作可能な度合いなので、上記、プロセスAiのプロセスAjに対する並列処理可能度λijとプロセスAjの実行頻度μjの関数として、例えば、次のように定義する。

【数３】

$$\lambda_i = \left(\sum_j \left((1/\mu_j)\cdot\lambda_{ij}\right)\right) / \sum_j (1/\mu_j)$$

こうして求めた並列動作度λiを処理時間ｔiと共に示したものが図５（ｂ）である。ここで、実行頻度μjが逆数（１回の実行時間となる）として扱われている理由は、プロセスの実行頻度が低いと他のプロセスが実行できる割合が高くなり、逆に、実行頻度が高いと他のプロセスが実行できる割合が低くなるからである。

【００２３】図５（ｂ）において、並列動作度λiから以下の如き意味を持たせる。プロセスAiが並列度大とは、λiがしきい値ρ以上となることを云う。プロセスAiが並列度小とは、λiがしきい値ρ以下となることを云う。ρの値は、プロセスの並列動作度λiとプロセッサの台数ｍから以下のようにして決める。｛λ1、…、λn｝をλiの降順（小さい値順）に並べたものをλ1′≧…≧λn′とする時、この値が上位（１／ｍ）・１００（％）に入るλ′が高速プロセッサで処理する対象となるので、上位（１／ｍ）・１００（％）に入るλ′の最小の値をρとする。本実施の形態では、ｍ＝４であるからρは上位２５％に入るλ′の最小値となる。

8

【００２４】図６は、プロセス割り付け判定プログラム１３の処理フローである。このプロセス割り付けとはプログラムの実際の処理を行う過程での動的な割り付けを指す。プロセスが実行される時、プロセス割り付け判定プログラム１３は、実行するプロセスをキュー（図６フロー６０１。いわゆる待行列のこと）より取得する。例えばキューの先頭にあるプロセスを取得する。この時、当該プロセスの並列動作度テーブル１４の情報（動作度の大小）を参照して（フロー６０３）並列度が小であれば高速プロセッサが空くのを待って、プロセスを高速プロセッサに割り付ける（フロー６０４）。並列度が大であれば、プロセッサ数分の並列度大のプロセス（本実施の形態では、４個）をキュー（待行列）より取り出し（フロー６０５～フロー６１１）、並列動作度テーブル１４の処理時間を参照して、実行させる並列度大のプロセスを処理時間順に並べ、全プロセッサが空くのを待って、処理時間が最長のプロセスを高速プロセッサへ、残りを高速でないプロセッサへ割り付ける（フロー６１２）。

【００２５】図７は、ある処理プログラム１１（メインプログラム）の概略構成図である。この図は、図６に基づき、逐次形と並列形とに割り付けた結果を示す図である。メインプログラム１１は、逐次処理プログラムＳ１と、この逐次処理プログラム終了後に実行される並列処理プログラムP1～P4と、この並列処理プログラム終了後に実行される逐次処理プログラムS2で構成されている。各プログラムS1、P1～P4、S2は、前述のプロセスAi（ｉ＝１～６）に相当する。ここでプロセスとは、プログラムの実行単位のことで、プログラムを実行するのに必要な情報（ＣＰＵが機械語の命令列として解釈できるテキストとデータ等）から成る。

【００２６】各プログラムS1、S2、P1～P4の並列処理可能度λijは、プログラムS1、S2が逐次プログラム、プログラムP1～P4が並列プログラムとした場合、以下のようになる（又は、なるものとする）。尚、λijの代わりに、印字の関係上λ（Ｓ、Ｐ）やλ（Ｐ、Ｐ）、λ（Ｐ、Ｓ）、λ（Ｓ、Ｓ）で示すことにする。当然にカッコ内の手前の英字がｉ、後ろの英字がｊを示す。

λ（S1、P1）＝λ（S1、P2）＝λ（S1、P3）＝λ

9

$(S_1, P_4) = \lambda(S_1, S_2) = 0$
$\lambda(P_1, S_1) = \lambda(P_1, S_2) = 0$
$\lambda(P_1, P_2) = \lambda(P_1, P_3) = \lambda(P_1, P_4) = 1$
$\lambda(P_3, S_1) = \lambda(P_2, S_2) = 0$
$\lambda(P_2, P_1) = \lambda(P_2, P_3) = \lambda(P_2, P_4) = 1$
$\lambda(P_3, S_1) = \lambda(P_3, S_2) = 0$
$\lambda(P_3, P_1) = \lambda(P_3, P_2) = \lambda(P_3, P_4) = 1$
$\lambda(P_4, S_1) = \lambda(P_4, S_2) = 0$
$\lambda(P_4, P_1) = \lambda(P_4, P_2) = \lambda(P_4, P_3) = 1$

また、各プログラムの実行頻度μは、下記の如くすべてのプロセッサで全て等しいとする。

$\mu(S_1) = \mu(P_1) = \mu(P_2) = \mu(P_3) = \mu(P_4) = \mu(S_2) = \mu_0$

このとき、各プログラムの並列動作度は、各々以下のようになる。

$\lambda(S_1) = \lambda(S_2) = 0$
$\lambda(P_1) = \lambda(P_2) = \lambda(P_3) = \lambda(P_4) = 4$

この場合のしきい値は、$\rho = 4$となる。従って、このプログラム１１は、並列動作度作成プログラムにより並列度小（$\lambda_i < \rho$）の逐次処理プログラムS_1、S_2と並列度大（$\lambda_i \geqq \rho$）の並列処理プログラムP_1～P_4に分類される。

【００２７】図８は、図１に示した４個のＣＰＵ１～４（ＣＰＵ１は高速）でメインプログラム１１を実行する時の、各プロセスＣＰＵへの配分を説明する図である。今、並列処理プログラムP_1～P_4の処理時間t_1～t_4を$t_3 \leqq t_1 \leqq t_2 \leqq t_4$とする。プロセス割り付け処理は、プロセスの並列度の大小と処理時間の長短により行うので高速プロセッサＣＰＵ１には、逐次処理プログラムS_1、S_2及び処理時間の長い並列処理プログラムP_4、プロセッサＣＰＵ２～ＣＰＵ４には、処理時間の短い並列処理プログラムP_1～P_3が割り付けられる。これにより、逐次処理と並列処理が混在するプログラムをそれを処理するのに適切なプロセッサに配分でき、全体として処理性を上げることができる。

【００２８】具体的な処理例としては、電力系統の計画支援業務がある。電力系統に送電線停止等の事故が起きたことを想定した場合、その復旧手順（系統のどのルートから回復したら良いか等）は複数の手段があり、計算機ではそれに対応して複数の復旧方針作成プログラム（例：系統の切り替えによる復旧、発電機の調整による復旧等）がある。このプログラムは前処理から計算に必要なデータをもらうと並列に処理できるので、

・逐次処理プログラム１（S_1）…前処理（事故の特定、復旧範囲の決定）
・並列処理プログラム１（P_1）…復旧方針作成プログラム１
・並列処理プログラム２（P_2）…復旧方針作成プログラム２
・並列処理プログラム３（P_3）…復旧方針作成プログラム３
・並列処理プログラム４（P_4）…復旧方針作成プログラム４
・逐次処理プログラム２（S_2）…後処理（結果の比較、評価）

となる。４つの復旧方針作成プログラムの中では、処理の複雑なものが処理時間がかかるので、この処理が高速プロセッサに割り当てられる。

【００２９】図９は、メインプログラム１１を従来の構成のＣＰＵ（４個の同じ性能のＣＰＵ）で従来手法のプロセス配分を行った場合を示す。従来手法は、ＣＰＵの性能が同じなので逐次処理S_1及びS_2がＣＰＵ１で動作している間、他のＣＰＵ２～ＣＰＵ４の空き時間が逐次用の高速プロセッサを使用する構成に比べて大きい。また、並列処理プログラムP_1～P_4の実行においてもＣＰＵ性能が同じため、並列処理時間のばらつきを考慮していないので、ＣＰＵ資源の無駄が生じる。この現象は、逐次処理の処理時間が長いほど、又、並列処理時間のばらつきが大きいほど顕著になる。

【００３０】尚、上記実施の形態では、高速１台、低速複数台としたが、高速２台以上の例での配分例へも拡張できる。また、低速複数台は同一機構又は同一構成としたが、そうでない例もあり、そうした能力に応じた割り付けを行ってもよい。また、逐次を高速、並列を低速に割り当てるものとしたが、低速と高速との区分をせずに、逐次を１台、並列を残りのものに割り当てる如きやり方もある。

【００３１】

【発明の効果】本発明によれば、従来の逐次型言語のプログラムをそのまま使用してプロセス間の並列処理を容易に実現し、逐次処理と並列処理が混在するプログラムを高速プロセッサと並列処理用の複数プロセッサの２種類のハード資源に有効に配分でき、トータルとして効率的なマルチプロセッサシステムができるという効果がある。

【図面の簡単な説明】

【図１】本発明のマルチプロセッサシステムの構成例図である。
【図２】並列動作度作成プログラムの動作を含む計算モードのフローチャート図である。
【図３】プロセス実行頻度作成プログラムでの処理フロー及びプロセス実行頻度テーブルを示す図である。
【図４】並列動作度テーブル作成プログラムの処理フロー例図である。
【図５】並列処理可能度テーブル、並列動作度テーブルのデータ例図である。
【図６】プロセス割り付け処理フロー図である。
【図７】プログラム割り付け例図である。
【図８】ＣＰＵへのプログラム割り付け例図である。
【図９】従来例によるプログラム割り付け例図である。

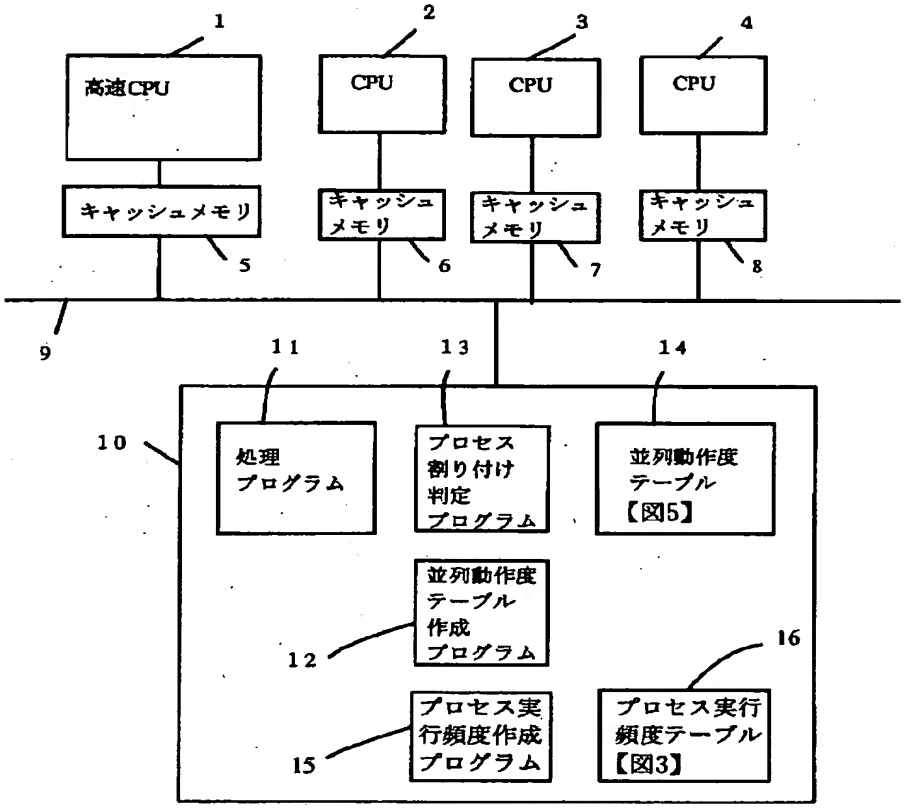
11

【符号の説明】
1　高速ＣＰＵ
2、3、4　低速ＣＰＵ
5、6、7、8　キャッシュメモリ
9　共通バス
１０　共有メモリ

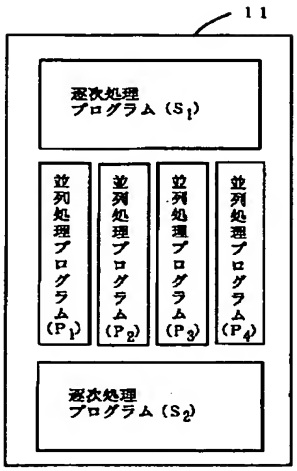
12

＊１１　処理プログラム
１２　並列動作度テーブル作成プログラム
１３　プロセス割り付け判定テーブル
１４　並列動作度
１５　プロセス実行頻度作成プログラム
＊　１６　プロセス実行頻度テーブル

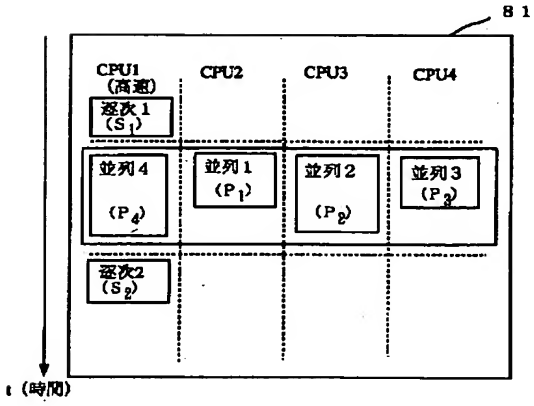
【図１】



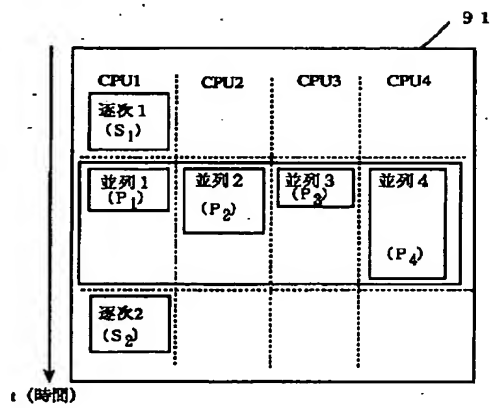
【図７】



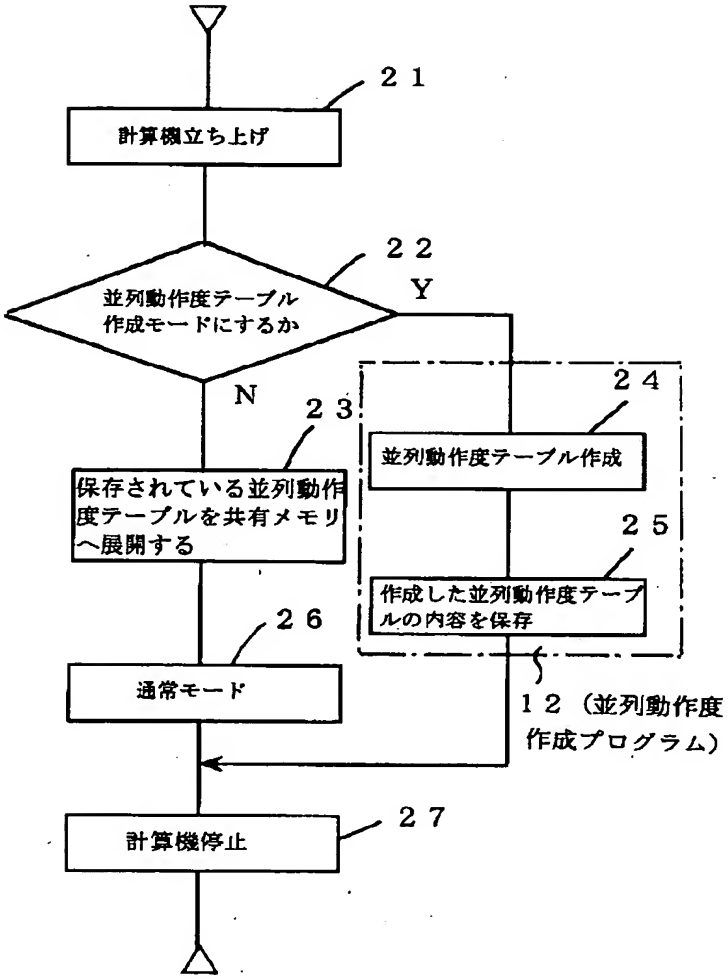
【図８】



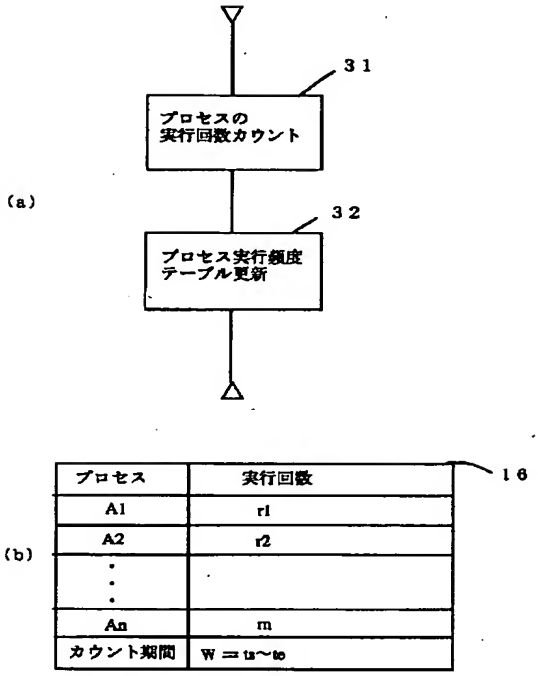
【図９】



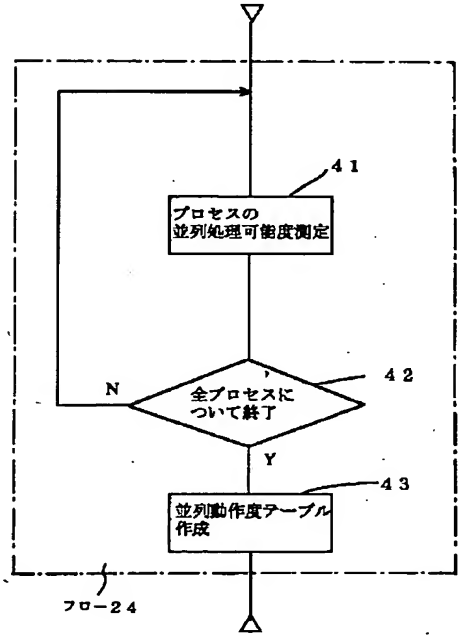
【図２】


２１
計算機立ち上げ
２２
並列動作度テーブル作成モードにするか
Y
N
２３
保存されている並列動作度テーブルを共有メモリへ展開する
２４
並列動作度テーブル作成
２５
作成した並列動作度テーブルの内容を保存
２６
通常モード
１２（並列動作度作成プログラム）
２７
計算機停止

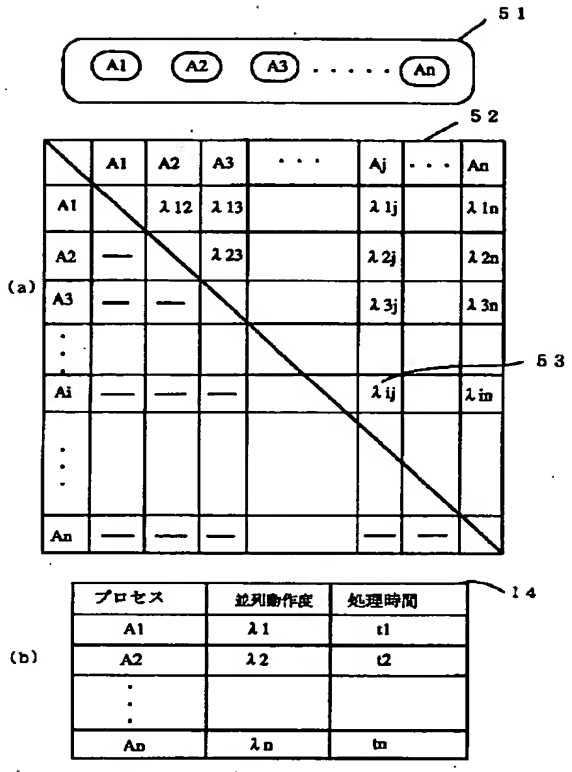
【図３】


(a)
３１
プロセスの実行回数カウント
３２
プロセス実行頻度テーブル更新
(b)
１６
プロセス
実行回数
A1
r1
A2
r2
An
m
カウント期間
W＝ts～te

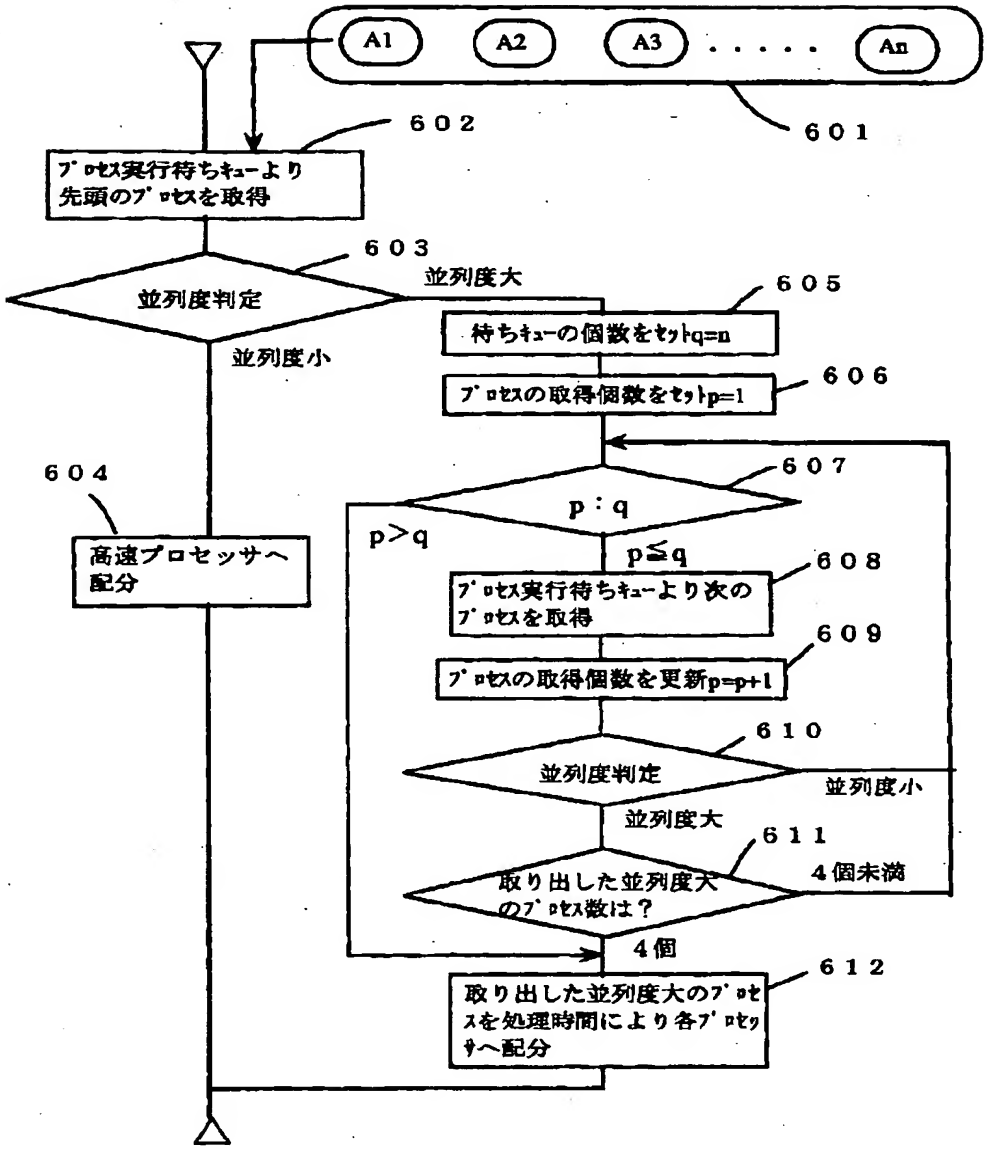
【図４】


４１
プロセスの並列処理可能度測定
４２
全プロセスについて終了
N
Y
４３
並列動作度テーブル作成
フロー２４

【図５】


51
A1
A2
A3
An
52
(a)
A1 A2 A3 Aj An
A1 λ12 λ13 λ1j λ1n
A2 λ23 λ2j λ2n
A3 λ3j λ3n
Ai λij λin
An
53
(b)
プロセス 並列動作度 処理時間
A1 λ1 t1
A2 λ2 t2
An λn tn
14

【図６】


A1
A2
A3
An
601
602
プロセス実行待ちキューより先頭のプロセスを取得
603
並列度判定
並列度大
並列度小
604
高速プロセッサへ配分
605
待ちキューの個数をセットq=n
606
プロセスの取得個数をセットp=1
607
p：q
p>q
p≦q
608
プロセス実行待ちキューより次のプロセスを取得
609
プロセスの取得個数を更新p=p+1
610
並列度判定
並列度小
並列度大
611
取り出した並列度大のプロセス数は？
4個未満
4個
612
取り出した並列度大のプロセスを処理時間により各プロセッサへ配分